AD-764 275

A MACHINE-INDEPENDENT ALGOL PROCEDURE FOR ACCURATE FLOATING-POINT SUMMATION

Michael A. Malcolm

Stanford University

Prepared for:

Office of Naval Research
National Science Foundation

June 1973

DISTRIBUTED BY:

NTIS
National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

A MACHINE-INDEPENDENT ALGOL PROCEDURE FOR ACCURATE FLOATING-POINT SUMMATION

by

Michael A. Malcolm

STAN-CS-73-374
June 1973

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

A MACHINE-INDEPENDENT ALGOL PROCEDURE
FOR ACCURATE FLOATING-POINT SUMMATION

Michael A. Malcolm

June 1973

This work was supported by the Office of Naval Research, Contract N00014-67-A-0112-0029, and NSF Contract GJ29988X.

procedure sum (x, n, m, result, fail);

value n, m; **integer** n, m; **real** result;

array x; **label** fail;

begin **comment** This Algol 60 procedure is an implementation of the floating-point summation technique described in Malcolm (1971). This implementation is machine-independent in the sense that it will work on any computer having a floating-point number system F characterized as follows: Each number $x \in F$ has a radix-β t-digit *fraction* where $t \geq 1$. The radix β can be any positive integer greater than 1. The exponent e is assumed to lie in the range

$$b \leq e \leq B ,$$

where $b \leq 0$ and $B > t$. Each nonzero $x \in F$ has the representation

$$x = \pm .d_1 d_2 \ldots d_t \cdot \beta^e ,$$

where $d_1, \ldots , d_t$ are integers satisfying

$$0 \leq d_i \leq \beta - 1 , \quad (i=1, \ldots ,t) .$$

The number 0 is contained in F, but no assumption is made about its representation. All floating-point operations (e.g., addition and multiplication) are assumed to result in either 0 or a *normalized* floating-point number contained in F. The machine may do either proper rounding or chopping (truncation). (Note that this definition of F excludes machines using extra-length accumulators for intermediate arithmetic. However, this algorithm is seldom needed on such machines.)

The parameters β and t of F are automatically computed at execution time by a technique described in Malcolm (1972). Since the range of the floating-point exponent cannot be determined automatically,

the input parameter m is used for allocating the set of accumulators used by the algorithm.

Provided no overflow or underflow occurs, and none of the x[i] are larger than 10^m, or smaller than 10^{-m}, in magnitude, and $n \leq \beta^{\ell+1}/16$, where $\ell = \lfloor t/2 \rfloor$, then

$$\text{result} \approx \sum_{i=1}^{n} x[i]$$

is returned with nearly full-precision accuracy. The bound on the relative error is given by Theorem 2 in Malcolm (1971) as

$$\lceil (t+1)/\lfloor \log_\beta 16 \rfloor \rceil \, \beta^{1-t} .$$

If any of the x[i] are larger than 10^m or smaller than 10^{-m}, then the error exit fail is taken. ;

```
Boolean rnd; integer beta, t, t2, nu, L, U;
procedure ENVRON (beta, t, rnd);
Boolean rnd; integer beta, t;
begin comment This procedure is an Algol 60 translation of the (first)
Fortran subroutine ENVRON given in Malcolm (1972). ;
      real a, b, e;

      for e := 2, 2×e while (a+1)-a=1 do a := e;
      for e := 2, 2×e while a+b=a do b := e;
      beta := (a+b)-a; rnd := a+(beta-1) > a; t := 0;
      for a := 1, beta×a while (a+1)-a=1 do t := t+1

end ENVRON;

ENVRON (beta, t, rnd); t2 := t÷2 ; nu := ℓn(16)/ℓn(beta);
U := entier (m×ℓn(10)/(ℓn(beta)×nu)) + 1;
L := entier((-m×ℓn(10)/ℓn(beta) - t2)/nu);
```

comment In the notation of Malcolm (1971), ℓ = t2 is the padding that each of the numbers added to the accumulators will have. Each of the x[i] will be split into two halves (i.e. q=2) having the last t2 digits equal zero. The variable nu above is used for ν defined in Equation (2) of Malcolm (1971). The value for nu computed above is rather arbitrary and was chosen to make nu sufficiently smaller than t2 . The variables U and L are the upper and lower bounds on the indices of the accumulators which are declared in the following block. They are chosen to allow the x[i] to range from 10^{-m} to 10^{m} in magnitude. In slightly different notation, they are

$$U = \lceil m/(\nu \times \ell og_{10}\beta) \rceil ,$$

$$L = \lfloor (-m/\ell og_{10}\beta - \lfloor t/2 \rfloor)/\nu \rfloor ;$$

```
begin array accumulators[L:U]; integer ex;
      real xL, xH;

      integer procedure e(x);
      value x; real x;
      begin comment This procedure computes the exponent e of the
          floating-point number x . ;
          real y, q; integer ex;
          x := abs(x); ex := 0; for y := 1,q
          while x>y do begin ex := ex+1; q := beta×y; end;
          for y := q, y/beta while x<y do ex := ex-1;
          e := ex
      end e;
```

```
comment initialize the array of accumulators;
for i:=L step 1 until U do accumulators[i] := 0;
comment accumulate the nonzero x[i]s;
for i:=1 step 1 until n do if x[i]≠0 then
begin ex := e(x[i]);
    if entier(ex/nu)>U ∨ ex-t2<L×nu then go to fail;
    comment Now the x[i] is split into a high- and low-order
    part, xH and xL. The method used here is to add the proper
    power of β to x[i] to force it to preshift t2 digits
    to the right and then either truncate or round the last t2
    significant digits. Then the same power of β is subtracted
    to cause a post normalization which brings in t2 trailing
    zero digits. The resulting high-order part of x[i] is then
    subtracted from x[i] to produce the low-order part such
    that the sum of the high- and low- order parts is exactly
    equal to x[i]. This method of splitting a floating-point
    number into two halves is similar to that given by Dekker
    (1971). ;
    xH := beta↑(ex-1+t2); xH := (xH+x[i]) - xH;
    xL := x[i] - xH;
    comment xH and xL can now be added to the appropriate
    accumulators. ;
    accumulators[entier(ex/nu)] := xH;
    accumulators[entier((ex-t2)/nu)] := xL
end; comment Now sum the accumulators in decreasing order. ;
result := 0;
for i:=U step -1 until L do
```

```
            result := result + accumulators[i]
        end
    end  sum
```

References

1. Dekker, T.J. (1971), "A Floating-Point Technique for Extending the Available Precision," Numer. Math. 18, 224-242.

2. Malcolm, Michael A. (1971), "On accurate floating-point summation," Comm. ACM, Vol. 14, No. 11, November, 731-736.

3. Malcolm, Michael A. (1972), "Algorithms to reveal properties of floating-point arithmetic," Comm. ACM, Vol. 15, No. 11, November, 949-951.